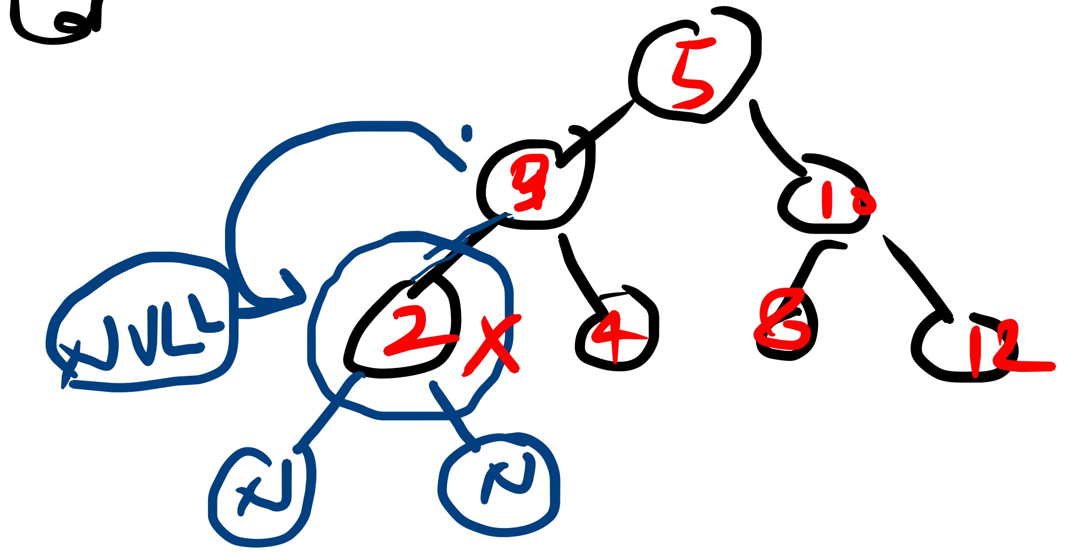
① Deletion from BST :-

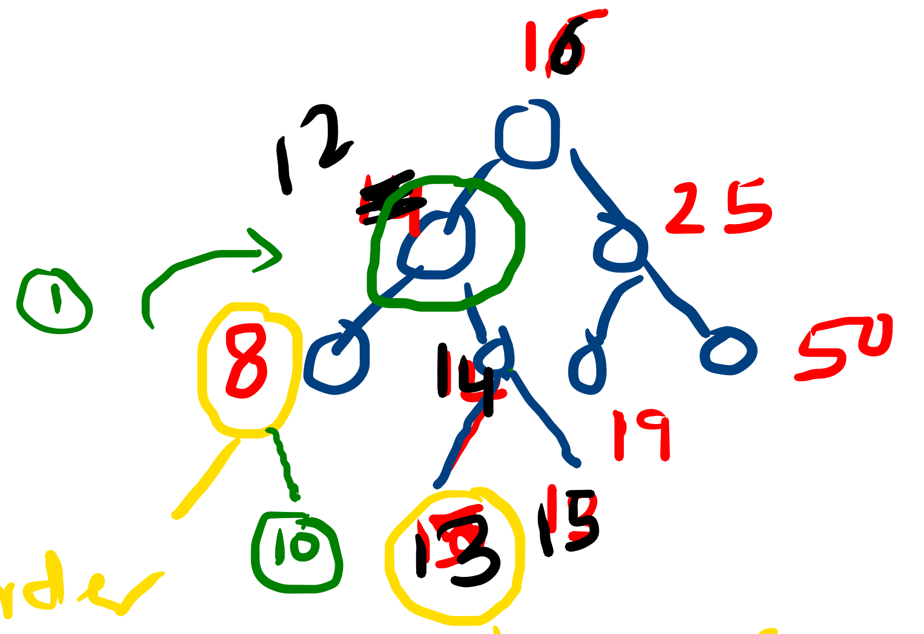
- Leaf Node < L, R => NULL → No child
- Node < L || R => NULL — Single child
- Between → Both child

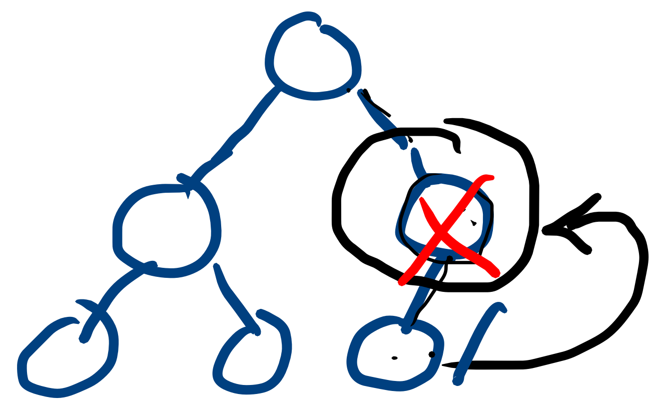


Case 1 — Delete ②

Case 2 →

Case 3 →





Inorder Predecessor

Inorder Successor

Go To

✓ In order successor → Key-Right → Extreme Left Node

✓ In order Predecessor → Key-Left → Extreme Right Node

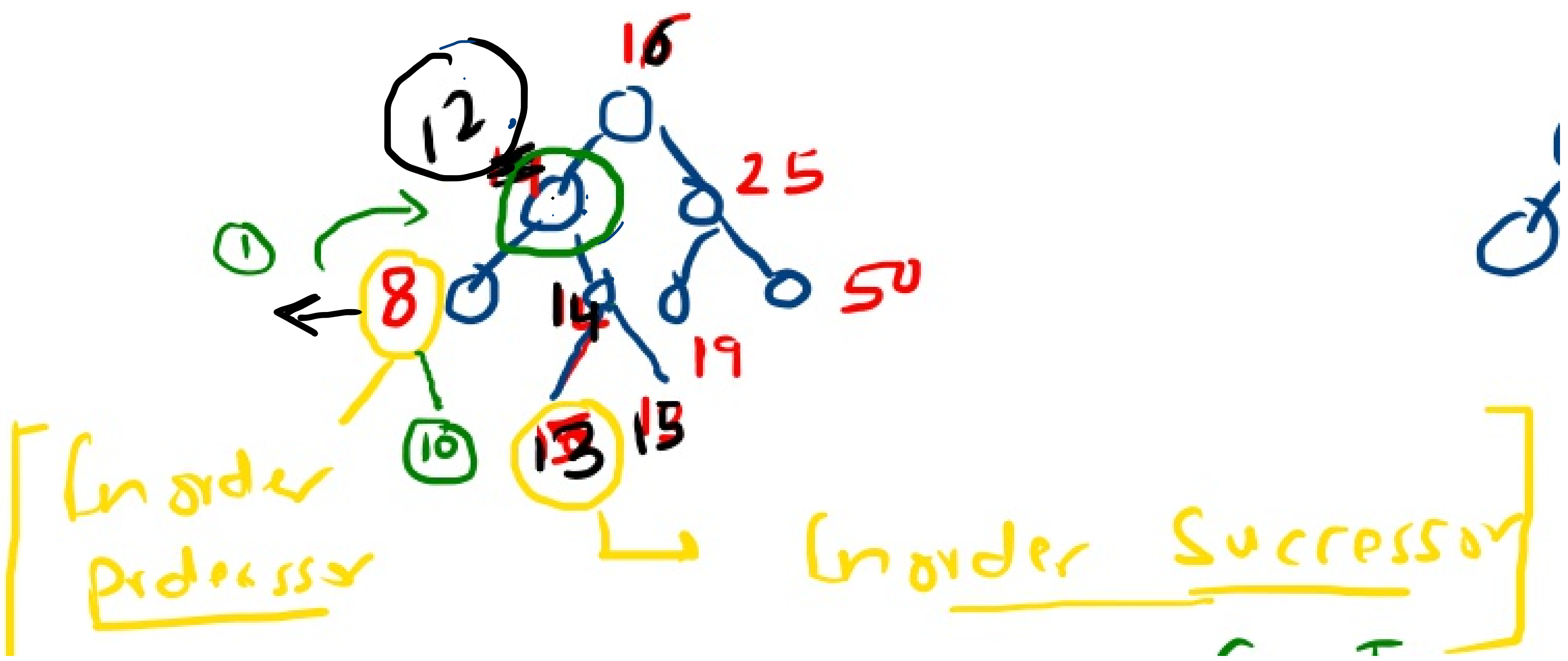
① Find the Inorder Successor of a Node!

iterative Code:

```
Node* Root;
int Key = 12;
```

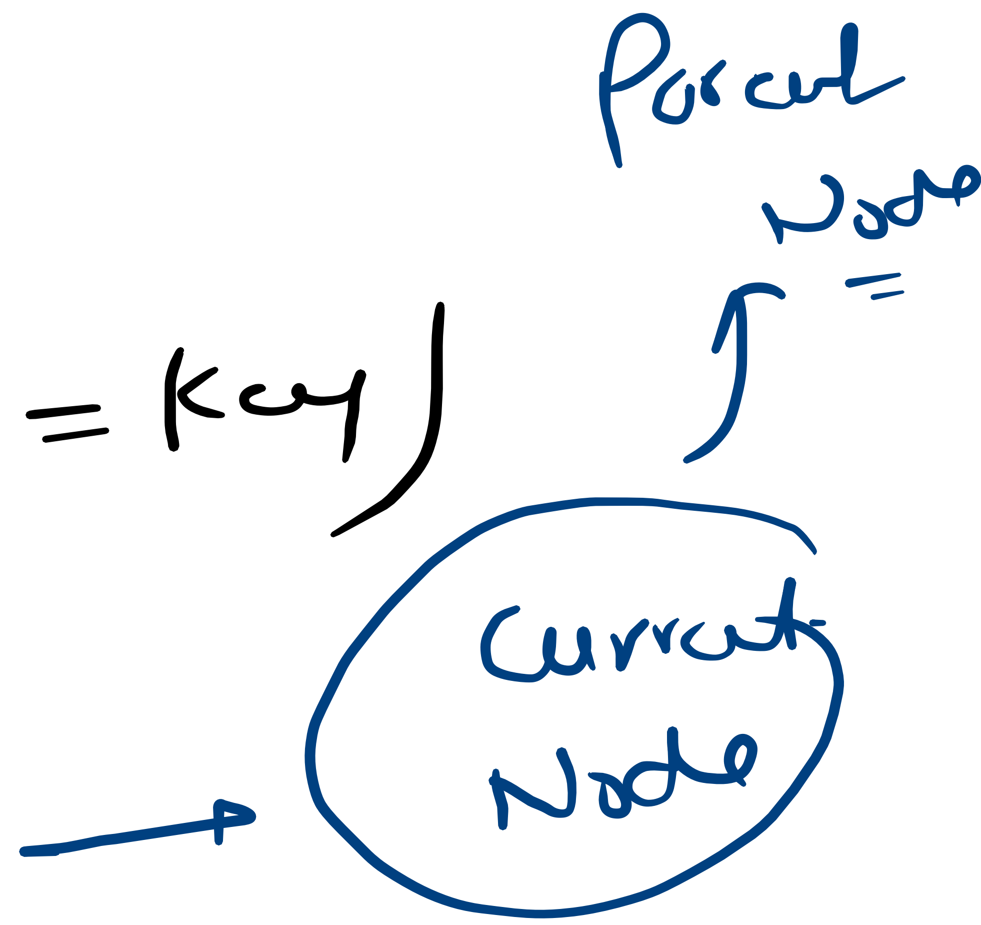
Node of 12

```
Node* InOS( Node* Root )
{
        RP = Root -> Right;
        while( RP -> Left != NULL )
            RP = RP -> Left;
        return( RP );
}
```



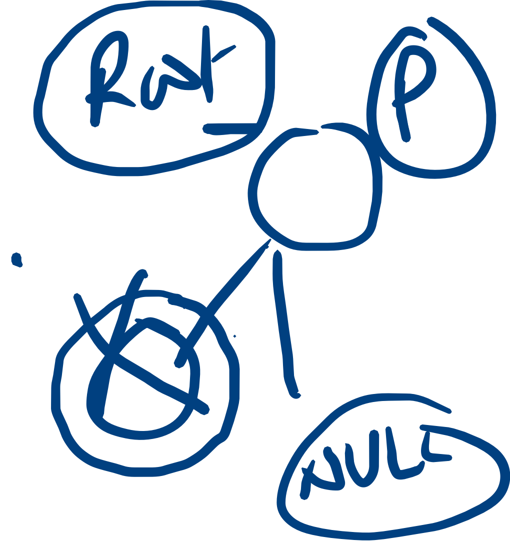
```
Node* Key Search( Node* Root, Key )
{
    while( Root -> value != Key )
    {
        if Key < Root -> value
            Root = Root -> Left;
        else
            Root -> Root -> Right;
    }
    return( Root );
```

```
① void DeleteN ( Node* Root , int Key)
  {
        Node* curr = Root;
          Node* Parent = NULL;
        while( curr != NULL && curr -> value != Key)
          {     Parent = curr;
                  if Key < current -> value)
                        curr = curr -> left;
                     else curr = curr -> Right;
           }
     if ( curr == NULL)
         Print( Key not Found) ; return;
```



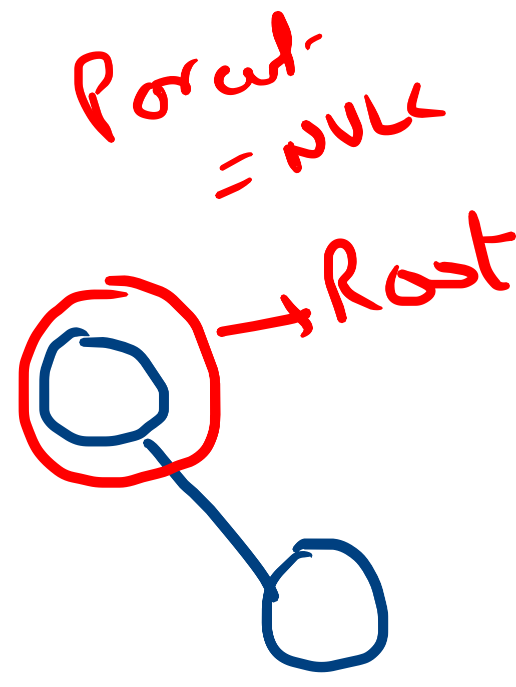
Case 1 → Having ~~Sh~~ Both the Children NULL

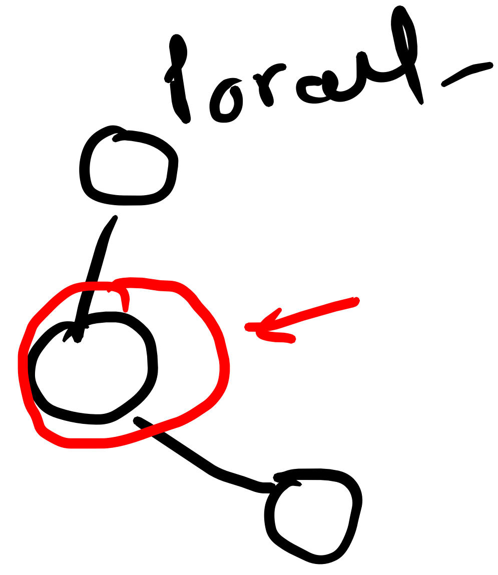
```
if (Curr → Left == NULL && Curr → Right == NULL)
{
    if (Root == Curr)
        Root = NULL;          ] — Corner Case Condition
    else
    {
        if (Curr == Parent → Left)
            Parent → Left = NULL;
        else
            Parent → Right = NULL
    }

    Free (Curr);
}
```



Case 2: Only one child:

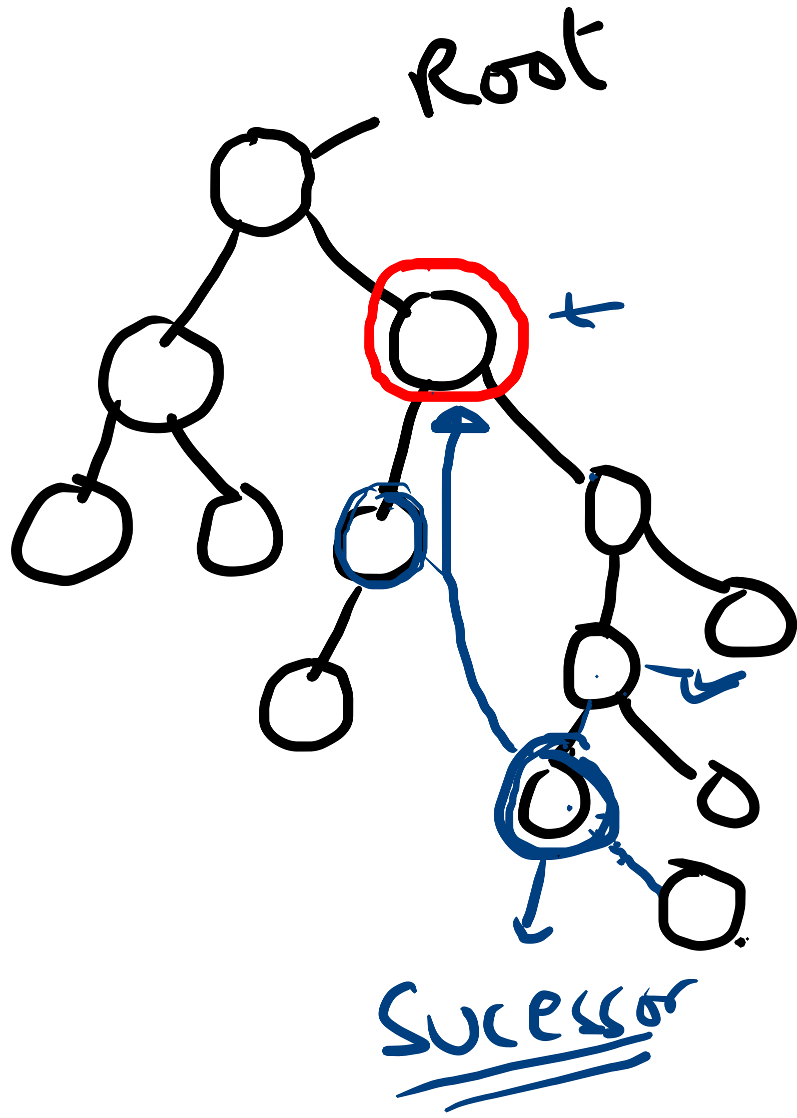
```
~~if~~ else if ( curr -> left == NULL || curr -> Right == NULL)
{
    Node* child = (curr -> left)? curr -> Left : curr -> Right;
    if (curr == Root)
    {   Root = child;
    }
    else
    {   if (curr == Parent -> Left)
            Parent -> Left = child;
        else Parent -> Right = child;
    }
    free (child);
}
```





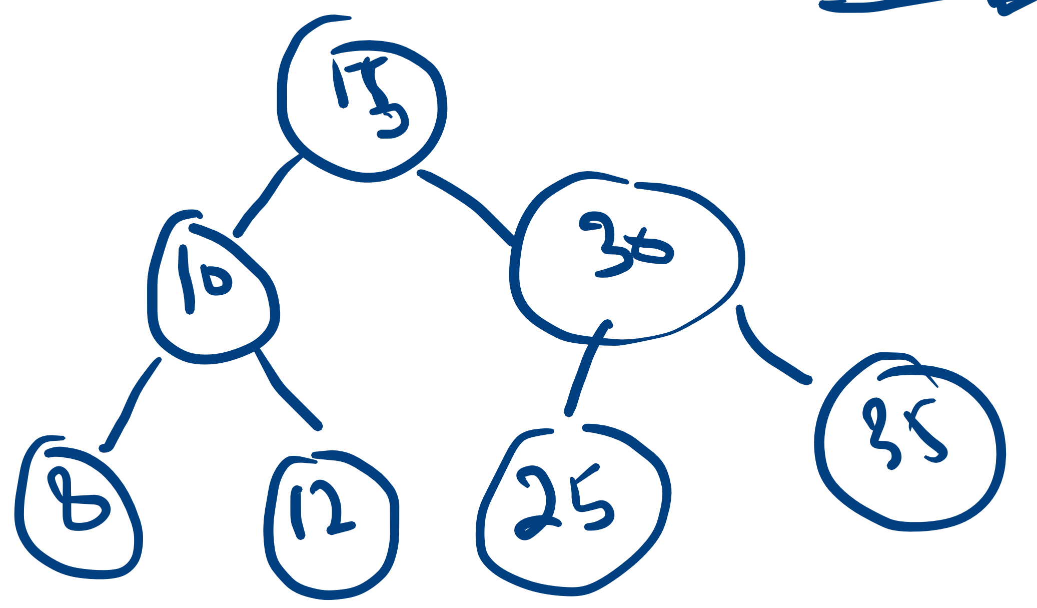
```
else {
    Node * SuccR = INOS( Current )|
    int Key = SuccR -> value;
    deleteN( Root , Key);
    Curr -> value = Key;
    }
}
```

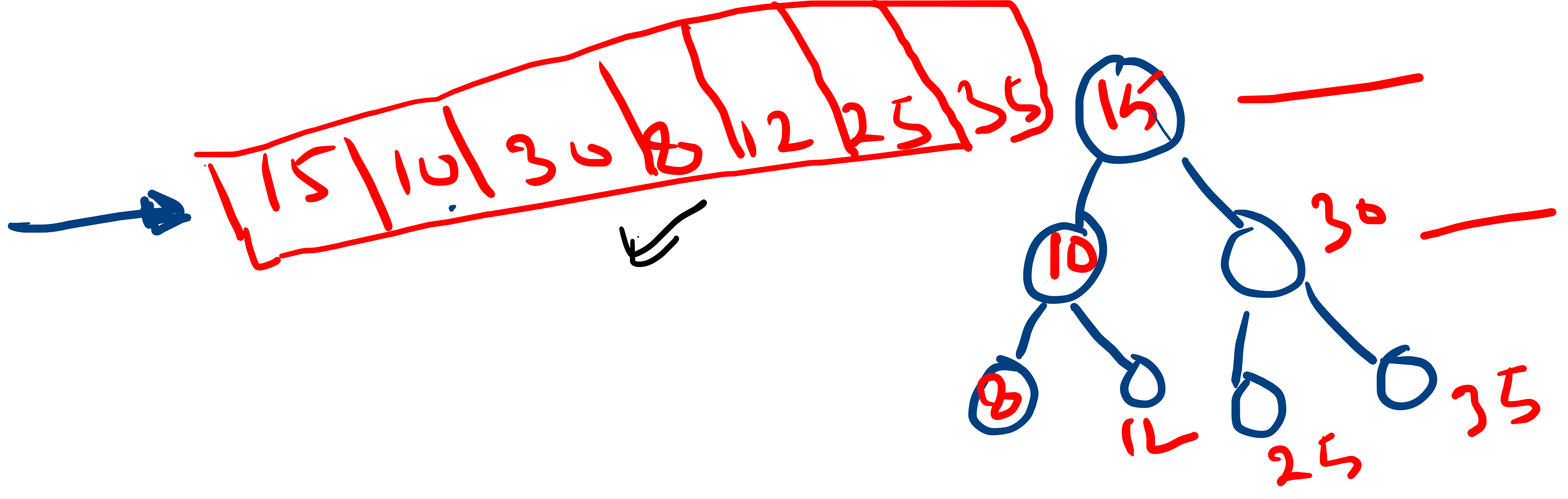
$\Longrightarrow$ $\boxed{\text{Code over}}$



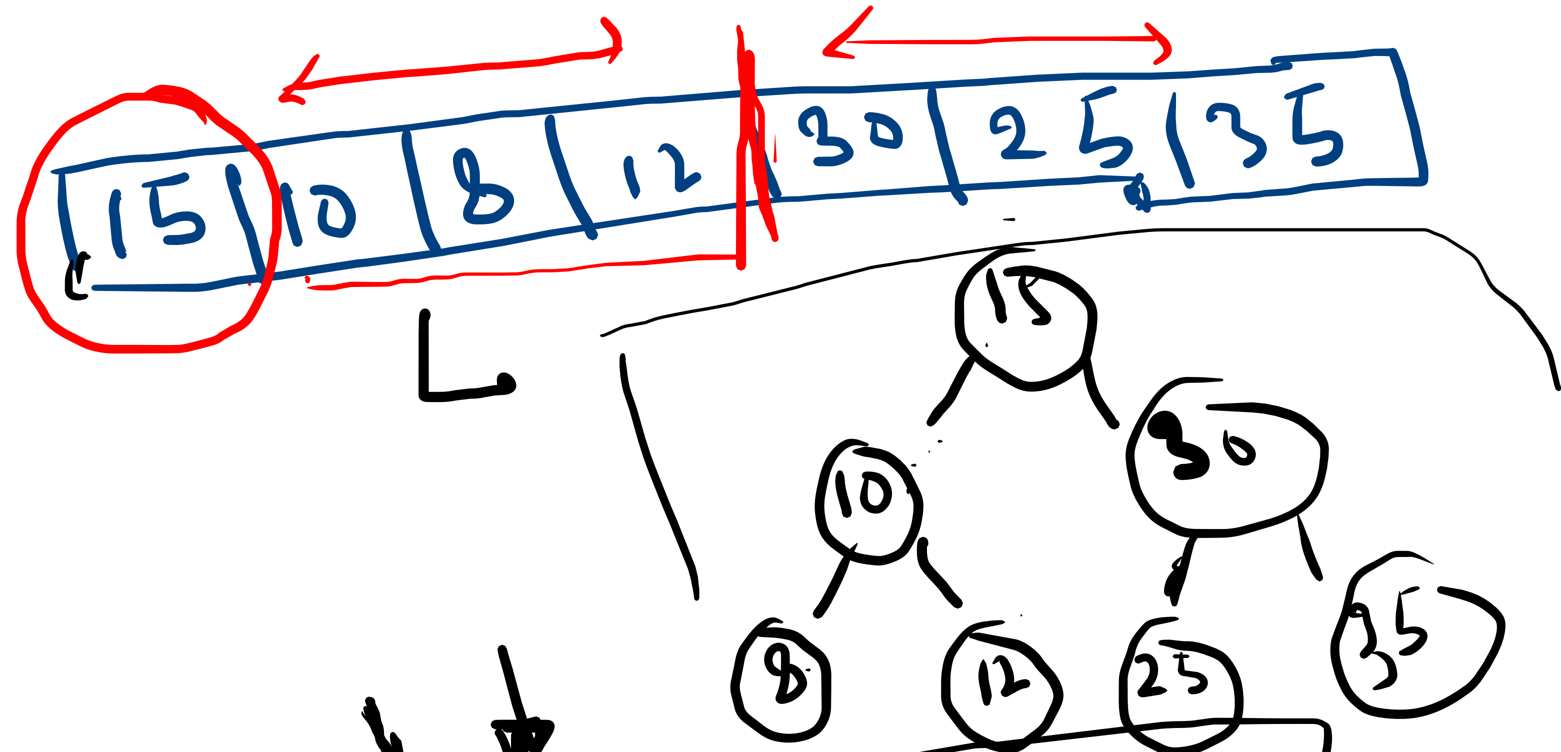
① BST from Level order Traversal:

→ BST insert:

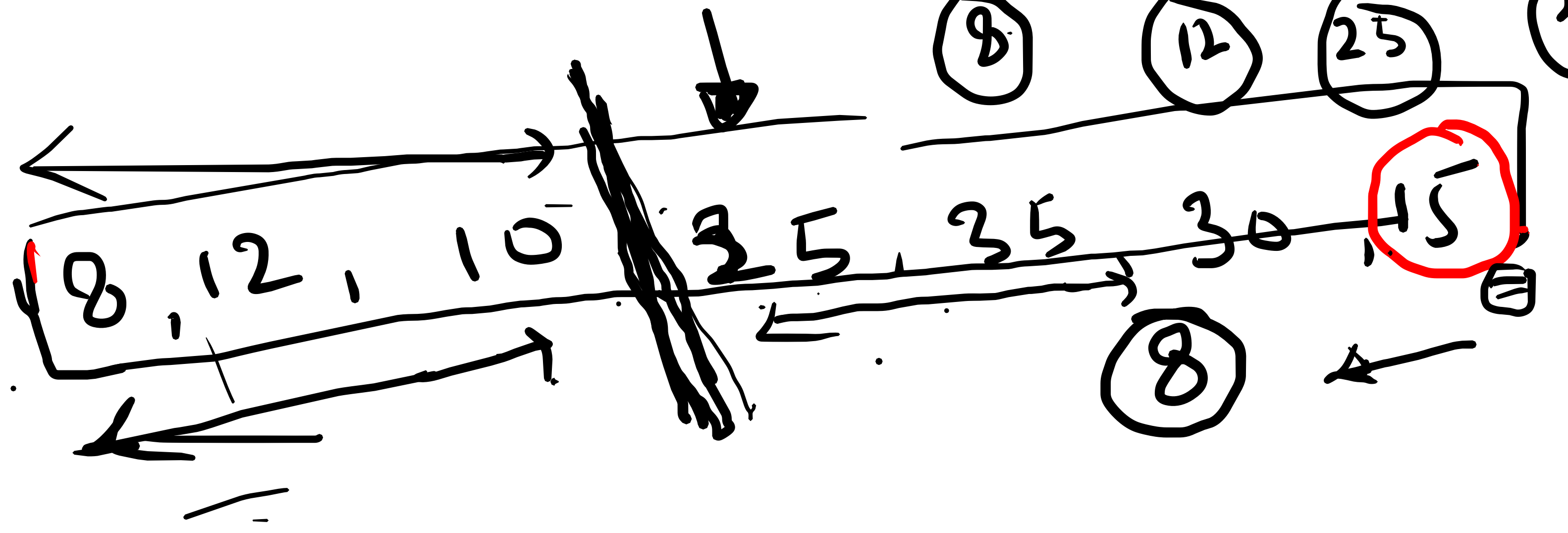




② BST from Pre order Traversal



③ BST ——— Post Order Traversal →



Last key = Root

Assignment:

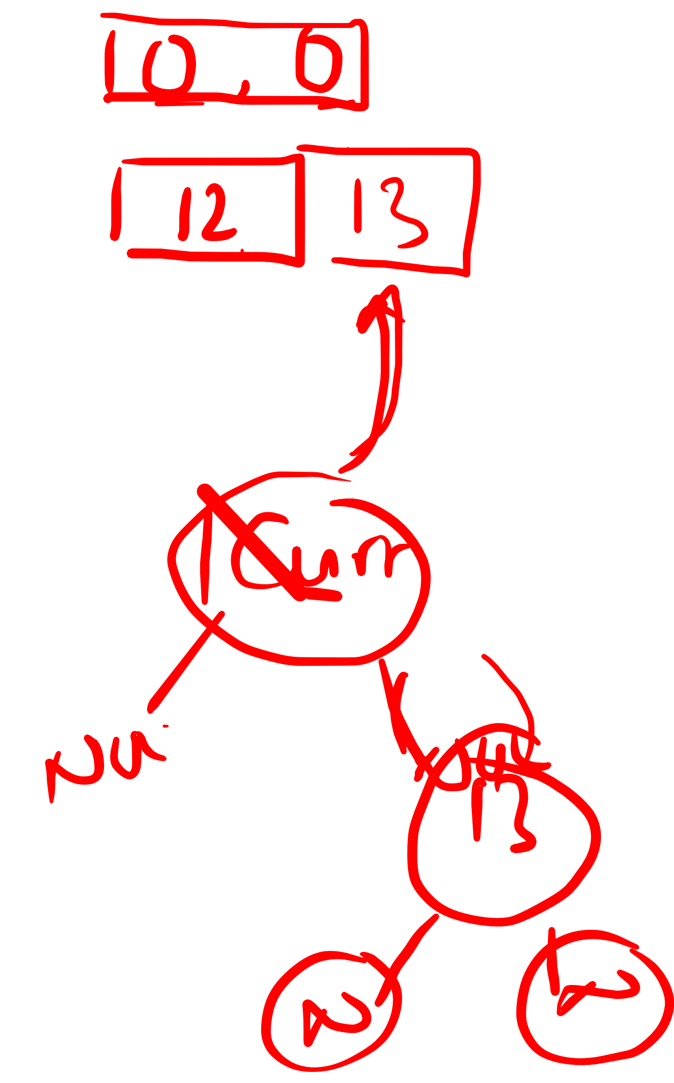
$\hookrightarrow$ $\boxed{\text{BST, BT}} \leftarrow$

Monday / Tue $\Rightarrow$ Heap

$\hookrightarrow$ Heavy =

③ BST From Post order Traversal:

```
int Arr = [✓];

    start = 0
    end = size(Arr)-1)

Node* Create BST( int Arr[], int start, int end)
{
   if (start > end) return NULL;

      Node* Curr = CreateNode( Arr[end]);
        int i=0;        for (i=0 ; i<end ; i++)
                        {     if Arr[i] > Arr[end];
                                   break;
                        }
        Curr -> Left = Create BST( Arr[] , start , i-1);
          Curr -> Right = Create BST ( Arr[] , i , end-1);
}    return (Curr);
```



0 , -1
0 , -1